Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS 2020)

Joint Inference of Reward Machines and Policies for Reinforcement Learning

Zhe Xu,[1*] Ivan Gavran,[2*] Yousef Ahmad,[1] Rupak Majumdar,[2] Daniel Neider,[2] Ufuk Topcu,[1] Bo Wu[1]

[1]University of Texas at Austin, Austin, TX
[2]Max Planck Institute for Software Systems, Kaiserslautern, Germany
{zhexu, utopcu, bwu3}@utexas.edu, {gavran, rupak, neider}@mpi-sws.org, ysa6549@gmail.com

Abstract

Incorporating *high-level knowledge* is an effective way to expedite reinforcement learning (RL), especially for complex tasks with sparse rewards. We investigate an RL problem where the high-level knowledge is in the form of *reward machines*, a type of Mealy machines that encode non-Markovian reward functions. We focus on a setting in which this knowledge is *a priori* not available to the learning agent. We develop an iterative algorithm that performs joint inference of reward machines and policies for RL (more specifically, q-learning). In each iteration, the algorithm maintains a *hypothesis* reward machine and a *sample* of RL episodes. It uses a separate q-function defined for each state of the current hypothesis reward machine to determine the policy and performs RL to update the q-functions. While performing RL, the algorithm updates the sample by adding RL episodes along which the obtained rewards are inconsistent with the rewards based on the current hypothesis reward machine. In the next iteration, the algorithm infers a new hypothesis reward machine from the updated sample. Based on an *equivalence* relation between states of reward machines, we transfer the q-functions between the hypothesis reward machines in consecutive iterations. We prove that the proposed algorithm converges almost surely to an optimal policy in the limit. The experiments show that learning high-level knowledge in the form of reward machines leads to fast convergence to optimal policies in RL, while the baseline RL methods fail to converge to optimal policies after a substantial number of training steps.

1 Introduction

In many reinforcement learning (RL) tasks, agents only obtain sparse rewards for complex behaviors over a long period of time. In such a setting, learning is very challenging and incorporating high-level knowledge can help the agent explore the environment in a more efficient manner (Taylor and Stone 2007). This high-level knowledge may be expressed as different levels of temporal or behavioral abstractions, or a hierarchy of abstractions (Nachum et al. 2018; Abel et al. 2018; Akrour et al. 2018).

The existing RL work exploiting the hierarchy of abstractions often falls into the category of hierarchical RL (Sutton, Precup, and Singh 1999; Dietterich 2000; Parr and Russell 1998). Generally speaking, hierarchical RL decomposes an RL problem into a hierarchy of subtasks, and uses a *meta-controller* to decide which subtask to perform and a *controller* to decide which action to take within a subtask (Barto and Mahadevan 2003).

For many complex tasks with sparse, non-Markovian rewards, there exists high-level knowledge capturing the temporal structure of the reward functions (Aksaray et al. 2016; Li, Vasile, and Belta 2017; Xu and Topcu 2019). One way of encoding a non-Markovian reward function and incorporating it into the standard q-learning algorithm (Watkins and Dayan 1992) is by using a type of Mealy machines named *reward machines* (Icarte et al. 2018). In many realistic situations, the reward machines are not straightforward to be handcrafted, as emphasized in (Toro Icarte et al. 2018). Oftentimes, such high-level knowledge is implicit and unknown to the learning agent.

In this paper, we provide a framework that enables an RL agent to infer high-level knowledge over its exploration process and utilize the inferred knowledge to effectively guide its future explorations. Specifically, we use reward machines to represent the high-level knowledge, and we present an iterative algorithm that performs joint inference of reward machines and policies (JIRP) for RL (more specifically, q-learning (Watkins and Dayan 1992)).

In each iteration, the JIRP algorithm maintains a *hypothesis reward machine* and a *sample* of RL episodes. It uses a separate q-function defined for each state of the current hypothesis reward machine to determine the policy and performs RL to update the q-functions. While performing RL, the algorithm updates the sample by adding *counterexamples* (i.e., RL episodes in which the rewards are inconsistent with the rewards based on the current hypothesis reward machine). The updated sample is used to infer a new *minimal* hypothesis reward machine, using automata learning techniques (Neider and Jansen 2013; Oncina and Garcia 1992). We prove that the proposed algorithm converges almost surely to an optimal policy in the limit if the maximal length of each RL episode is sufficiently long.

We use two algorithmic optimizations in the proposed algorithm for its efficient implementation. First, we batch the counterexamples and trigger the inference of a new hypothesis reward machine periodically. In this way, we can adjust

[*]The first two authors have contributed equally; the rest of the authors are ordered alphabetically.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the frequency of inferring new hypotheses reward machines. Second, we utilize the experiences from previous iterations by transferring the q-functions between *equivalent* states of two hypothesis reward machines in consecutive iterations. We prove that the optimal convergence guarantees are preserved with the two algorithmic optimizations.

We implement the proposed JIRP approach and compare it with three baseline methods: q-learning in *augmented state space*, hierarchical reinforcement learning (Kulkarni et al. 2016), and deep reinforcement learning with double q-learning (Hasselt, Guez, and Silver 2016). We evaluated them in three scenarios: an autonomous vehicle scenario, an office world scenario, and a Minecraft world scenario, running a set of tasks for every scenario. The experiments show that the proposed JIRP approach significantly outperforms the three baseline algorithms in all three scenarios.

Related work Our work is closely related to the use of formal methods in RL, such as RL for reward machines (Icarte et al. 2018) and RL with temporal logic specifications (Aksaray et al. 2016; Li, Vasile, and Belta 2017; Fu and Topcu 2014; Toro Icarte et al. 2018; Wen, Papusha, and Topcu 2017; Alshiekh et al. 2018). For example, the authors in (Icarte et al. 2018) develop a method called *q-learning for reward machines* (QRM) and show that QRM can converge almost surely to an optimal policy in the tabular case. Furthermore, QRM outperforms both q-learning and hierarchical RL for tasks where the reward functions can be encoded by reward machines. The results mentioned above mostly rely on the assumption that the high-level knowledge (e.g., reward machines) is given. In this work, we drop that assumption and infer reward machines from data.

With implicit high-level knowledge representing the reward functions, an approach related to this work has been recently proposed in (Xu and Topcu 2019), where the inferred high-level knowledge is represented by *temporal logic* formulas and the authors perform q-learning in the augmented state space constructed from the inferred temporal logic formulas. In comparison to temporal logic formulas, the reward machines used in this paper can express a larger set of non-Markovian reward functions. Besides, the method in (Xu and Topcu 2019) is mainly used for transfer learning.

In recent work (Icarte et al. 2019), the authors propose a method, called LRM, to learn reward machines that represent the memory of partially observable Markov decision processes (POMDP) and perform RL with the learned reward machines. While superficially similar, the two approaches are useful in different contexts. JIRP excels when the temporal structure of the rewards contains all the useful information because it learns a smallest and complete representation of the rewards. In this situation, LRM tends to learn larger-than-necessary reward machines, which makes reinforcement learning slower. In cases where the reward function is Markovian, on the other hand, JIRP is not as useful, while LRM can still speed up RL as it can infer useful information about the structure of the POMDP that is not contained in the reward function.

Learning high-level knowledge about rewards from experiences is also used in inverse RL (Arora and Doshi 2018). In contrast to the inverse RL where rewards are learned from demonstrations generated by human experts, JIRP learns reward machines from the agent's exploration of the environment. Function approximation is often used in inverse RL literature (Wulfmeier, Ondruska, and Posner 2015; Wen, Papusha, and Topcu 2017). For non-Markovian rewards, it is not known beforehand how much history is needed for expressing the rewards. Our approach provides a systematic way to determine the structure of the reward and performs RL with optimal convergence guarantees.

Motivating Example

As a motivating example, let us consider an autonomous vehicle navigating a residential area, as sketched in Figure 1. As is common in many countries, some of the roads are priority roads. While traveling on a priority road, a car has the right-of-way and does not need to stop at intersections. In the example of Figure 1, all the horizontal roads are priority roads (indicated by gray shading), whereas the vertical roads are ordinary roads.

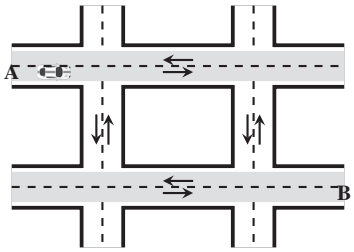


Figure 1: Map of a residential area.

Let us assume that the task of the autonomous vehicle is to drive from position "A" on the map to position "B" while obeying the traffic rules. To simplify matters, we are here only interested in the traffic rules concerning the right-of-way and how the vehicle acts at intersections with respect to the traffic from the intersecting roads. Moreover, we make the following two further simplifications: (1) the vehicle correctly senses whether it is on a priority road and (2) the vehicle always stays in the road and goes straight forward while not at the intersections.

The vehicle is obeying the traffic rules if and only if

- it is traveling on an ordinary road and stops for exactly one time unit at the intersections;
- it is traveling on a priority road and does not stop at the intersections.

We intend to achieve the above task in the setting of episodic RL. Specifically, after each episode of 100 time units, the vehicle receives a reward of 1 if it reached B while obeying the traffic rules; otherwise it receives a reward of 0. It can be seen that the (implicit) reward function is non-Markovian (the reward depends not only on the current state, but also on the history of states from which one can decide if the traffic rules were obeyed).

2 Preliminaries

In this section we introduce necessary background on reinforcement learning and reward machines.

Markov Decision Processes and Reward Machines

Definition 1. *A* labeled Markov decision process (MDP) *is a tuple* $\mathcal{M} = (S, s_I, A, p, R, \gamma, \mathcal{P}, L)$ *consisting of a finite state space* S*, an agent's initial state* $s_I \in S$*, a finite set of actions* A*, and a probabilistic transition function* $p\colon S \times A \times S \to [0, 1]$*. A reward function* $R : (S \times A)^+ \times S \to \mathbb{R}$ *and a discount factor* $\gamma \in [0, 1)$ *together specify payoffs to the agent. Finally, a finite set* $\mathcal{P}$ *of propositional variables, and a labeling function* $L : S \times A \times S \to 2^{\mathcal{P}}$ *determine the set of relevant high-level events that the agent senses in the environment. We define the size of* $\mathcal{M}$*, denoted as* $|\mathcal{M}|$*, to be* $|S|$ *(i.e., the cardinality of the set* S*).*

Our definition of MDPs differs from the "usual" definition used in reinforcement learning (e.g., (Sutton and Barto 2018)) in two ways. First, the reward function R is defined over the whole history (i.e., the reward is *non-Markovian*). Second, Definition 1 includes a set $\mathcal{P}$ of propositions and a labeling function L assigning sets of propositions—also called *labels*—to each transition (s, a, s') of an MDP. Labels represent a high-level description of a transition that is sufficient to express a reward. They come from expert knowledge of what is relevant for successfully executing a task and are assumed to be available to the agent. In the motivating example with the autonomous vehicle, for instance, it is enough to reason about the vehicle's behavior while entering an intersection from a priority or an ordinary road (and these would be the propositional variables). Note, however, that the full state space is still necessary to capture the model's dynamics, which might vary across different elements of S.

A *policy* is a function mapping states in S to a probability distribution over actions in A. At state $s \in S$, an agent using policy π picks an action a with probability $\pi(s, a)$, and the new state s' is chosen with probability $p(s, a, s')$. A policy π and the initial state s_I together determine a stochastic process and we write $S_0 A_0 S_1 \dots$ for the random trajectory of states and actions.

A *trajectory* is a realization of this stochastic process: a sequence of states and actions $s_0 a_0 s_1 \dots s_k a_k s_{k+1}$, with $s_0 = s_I$. Its corresponding *label sequence* is $\ell_0 \ell_1 \dots \ell_k$ where $L(s_i, a_i, s_{i+1}) = \ell_i$ for each $i \leq k$. Similarly, the corresponding *reward sequence* is $r_0 r_1 \dots r_k$, where $r_i = R(s_0 a_0 \dots s_i a_i s_{i+1})$, for each $i \leq k$. We call the pair $(\lambda, \rho) := (\ell_0 \dots \ell_k, r_0 \dots r_k)$ a *trace*.

A trajectory $s_0 a_0 s_1 a_1 \dots s_k a_k s_{k+1}$ achieves a reward $\sum_{i=0}^{k} \gamma^i R(s_0 a_0 \dots s_i a_i s_{i+1})$. The objective of the agent is to maximize the expected cumulative reward, $\mathbb{E}_\pi[\sum_{i=0}^{\infty} \gamma^i R(S_0 A_0 \dots S_{i+1})]$.

Encoding a (non-Markovian) reward in a type of finite state-machine is achieved by reward machines (Icarte et al. 2018; Camacho et al. 2019). [1] Technically, a reward machine is a special instance of a Mealy machine (Shallit 2008), the one that has real numbers as its output alphabet and subsets of propositional variables (originating from an underlying MDP) as its input alphabet. (To accentuate this connection in the following definition, we explicitly mention both the input alphabet $2^{\mathcal{P}}$ and the output alphabet $\mathbb{R}$.)

Definition 2. *A* reward machine $\mathcal{A} = (V, v_I, 2^{\mathcal{P}}, \mathbb{R}, \delta, \sigma)$ *consists of a finite, nonempty set* V *of states, an initial state* $v_I \in V$*, an input alphabet* $2^{\mathcal{P}}$*, an output alphabet* $\mathbb{R}$*, a (deterministic) transition function* $\delta\colon V \times 2^{\mathcal{P}} \to V$*, and an output function* $\sigma\colon V \times 2^{\mathcal{P}} \to \mathbb{R}$*. We define the size of* $\mathcal{A}$*, denoted as* $|\mathcal{A}|$*, to be* $|V|$ *(i.e., the cardinality of the set* V*).*

When a reward machine is in one of its finitely many states, upon reading a label, it outputs a reward and transitions into a next state. Formally, the run of a reward machine $\mathcal{A}$ on a sequence of labels $\ell_0 \dots \ell_k \in (2^{\mathcal{P}})^*$ is a sequence $v_0(\ell_0, r_0)v_1(\ell_1, r_1) \dots v_k(\ell_k, r_k)v_{k+1}$ of states and label-reward pairs such that $v_0 = v_I$ and for all $i \in \{0, \dots, k\}$, we have $\delta(v_i, \ell_i) = v_{i+1}$ and $\sigma(v_i, \ell_i) = r_i$. We write $\mathcal{A}(\ell_0 \dots \ell_k) = r_0 \dots r_k$ to connect the input label sequence to the sequence of rewards produced by the machine $\mathcal{A}$. We say that a reward machine $\mathcal{A}$ *encodes* the reward function R of an MDP if for every trajectory $s_0 a_0 \dots s_k a_k s_{k+1}$ and the corresponding label sequence $\ell_0 \dots \ell_k$, the reward sequence equals $\mathcal{A}(\ell_0 \dots \ell_k)$.

An interesting (and practically relevant) subclass of reward machines is given by Mealy machines with a specially marked subset of *final states*, the output alphabet $\{0, 1\}$, and the output function mapping a transition to 1 if and only if the end-state is a final state and the transition is not a self-loop. Additionally, final states must not be a part of any cycle, except for a self-loop. This special case can be used in reinforcement learning scenarios with *sparse* reward functions (e.g., see the reward machines used in the case studies in (Icarte et al. 2018)).

Figure 2 shows a reward machine for our motivating example, where transitions not shown in the figure are self-loops with reward 0. Intuitively, state v_0 corresponds to the vehicle traveling on a priority road, while v_1 and v_2 correspond to the vehicle traveling and stopped on an ordinary road, respectively. While in v_0, the vehicle ends up in a sink state v_3 if it stops at an intersection (labels for which the propositional formula sp is true). While in state v_1, the vehicle gets to the sink state v_3 if it does not stop at an intersection ($\neg sp$), and gets to state v_2 if it stops at an intersection (sp). While in state v_2, the vehicle gets to the sink state v_3 if it stops again at the same intersection (sp); it gets back to state v_0 if it enters a priority road ($\neg sp \wedge pr$); and it gets back to state v_0 if it keeps moving on an ordinary road ($\neg sp \wedge \neg pr$). The reward machine switches among states v_0, v_1 and v_2 if the vehicle is obeying the traffic rules. Finally, a reward of 1 is obtained if from v_0 the goal position B is reached.

Reinforcement Learning With Reward Machines

In reinforcement learning, an agent explores the environment modeled by an MDP, receiving occasional rewards according to the underlying reward function (Sutton and Barto

[1] The reward machines we are using are the so-called *simple reward machines* in the parlance of (Icarte et al. 2018), where every output symbol is a real number.

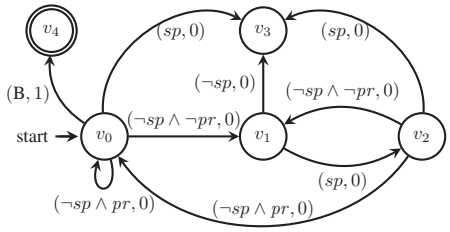


Figure 2: Reward machine for the autonomous vehicle. sp: stops at an intersection; pr: ends up in a priority road. An edge $(\neg sp \wedge \neg pr, 0)$ between v_0 and v_1 means that the reward machine will transition from v_0 to v_1 if the propositional formula $\neg sp \wedge \neg pr$ is true for a sensed label and it will output a reward equal to zero.

Algorithm 1: QRM_episode

1 **Hyperparameter**: episode length *eplength*, learning rate α, discount factor γ
2 **Input:** a reward machine $(V, v_I, 2^{\mathcal{P}}, \mathbb{R}, \delta, \sigma)$, a set of q-functions $Q = \{q^v | v \in V\}$
3 $s \leftarrow \mathit{InitialState}(); v \leftarrow v_I; \lambda \leftarrow []; \rho \leftarrow []$
4 **for** $0 \leq t < \mathit{eplength}$ **do**
5 $\quad a \leftarrow \text{GetEpsilonGreedyAction}(q^v, s)$
6 $\quad s' \leftarrow \text{ExecuteAction}(s, a)$
7 $\quad v' \leftarrow \delta(v, L(s, a, s'))$
8 $\quad r \leftarrow \sigma(v, L(s, a, s'))$ or observe reward in JIRP
9 $\quad q^v(s, a) \leftarrow$
$\quad (1 - \alpha) \cdot q^v(s, a) + \alpha \cdot (r + \gamma \cdot \max_a q^v(s', a))$
10 $\quad$ **for** $\hat{v} \in V \setminus \{v\}$ **do**
11 $\quad\quad \hat{v}' \leftarrow \delta(\hat{v}, L(s, a, s'))$
12 $\quad\quad \hat{r} \leftarrow \sigma(\hat{v}, L(s, a, s'))$
13 $\quad\quad$ update $q^{\hat{v}}(s, a)$ using reward $\hat{r}$
14 $\quad$ append $L(s, a, s')$ to λ; append r to ρ
15 $\quad s \leftarrow s'; v \leftarrow v'$
16 **return** (λ, ρ, Q)

2018). One way to learn an optimal policy is tabular q-learning (Watkins and Dayan 1992). There, the value of the function $q(s, a)$, which represents the expected future reward for the agent taking action a in state s, is iteratively updated. For MDPs with a Markovian reward function, q-learning converges to an optimal policy in the limit provided that all state-action pairs are seen infinitely often (Watkins and Dayan 1992).

The QRM algorithm modifies q-learning to learn an optimal policy when the reward function is encoded by a reward machine (Icarte et al. 2018). Algorithm 1 shows one episode of the QRM algorithm (with episode length *eplength*, learning rate α, and discount factor γ as its hyperparameters and a reward machine as its input). It maintains a set Q of q-functions, denoted as q^v for each state v of the reward machine (their initial values can be specified as an input to the algorithm).

The current state v of the reward machine guides the exploration by determining which q-function is used to choose the next action (Line 5). However, in each single exploration step, the q-functions corresponding to all reward machine states are updated (Lines 9 and 13).

The modeling hypothesis of QRM is that the rewards are known, but the transition probabilities are unknown. When using QRM as a part of JIRP, this knowledge is not available and the rewards must be *observed* (see Line 8). During the execution of the episode, traces (λ, ρ) of the reward machine are collected (Line 14) and returned in the end. While not necessary for q-learning, the traces will be useful in our algorithm to check the consistency of an *inferred* reward machine with rewards received from the environment.

3 Joint Inference of Reward Machines and Policies (JIRP)

Given a reward machine, the QRM algorithm learns an optimal policy. In many situations, however, assuming the knowledge of the reward function (and thus the reward machine) is unrealistic, and even if the reward function is known, it can be very challenging to formalize it in terms of a reward machine. In this section, we describe an RL algorithm that *iteratively* infers (i.e., learns) a correct reward machine together with the optimal policy for a given RL problem and a labeling function, which provides high-level knowledge about relevant events.

Our algorithm combines an automaton learning algorithm to infer hypothesis reward machines and the QRM algorithm for RL on the current candidate. Inconsistencies between the hypothesis machine and the observed traces are used to trigger a re-learning of the reward machine. We show that the resulting iterative algorithm converges in the limit almost surely to the reward machine encoding the reward function and to an optimal policy for this reward machine.

JIRP Algorithm

Algorithm 2 describes our JIRP algorithm. It maintains a hypothesis reward machine $\mathcal{H}$ and runs the QRM algorithm to learn an optimal policy (given $\mathcal{H}$). The episodes of QRM are used to collect traces and update q-functions. As long as the traces are consistent with the current hypothesis reward machine $\mathcal{H}$, QRM interacts with the environment using $\mathcal{H}$ to guide the learning process. However, if a trace (λ, ρ) is detected that is inconsistent with the hypothesis reward machine (i.e., $\mathcal{H}(\lambda) \neq \rho$, Line 6), our algorithm records it in a set X (Line 7)—we call the trace (λ, ρ) a *counterexample* and the set X a *sample*. Every time the sample is updated, JIRP infers a *minimal* reward machine (Line 8) that is consistent with the sample (we formalize this shortly).

Note that JIRP infers not an arbitrary consistent reward machine but a *minimal* one (i.e., a consistent reward machine with the fewest number of states among all consistent reward machines). This additional requirement can be seen as an Occam's razor strategy (Löding, Madhusudan, and Neider 2016) and is crucial in that it enables JIRP to converge to the optimal policy in the limit.

Algorithm 2: JIRP

```
Initialize the hypothesis reward machine H with a set of states V
Initialize a set of q-functions Q = {q^v | v ∈ V}
Initialize X = ∅
for episode n = 1, 2, ... do
    (λ, ρ, Q) = QRM_episode(H, Q)
    if H(λ) ≠ ρ then
        add (λ, ρ) to X
        infer a new, minimal hypothesis reward machine H based on the traces in X
        re-initialize Q
```

Inference of Minimal Reward Machines

Intuitively, a sample $X \subset (2^{\mathcal{P}})^+ \times \mathbb{R}^+$ contains a finite number of counterexamples, and we would like to construct a (new) reward machine that is both *minimal* and *consistent with* X. This objective is formalized next.

Task 1. *Given a finite set $X \subset (2^{\mathcal{P}})^+ \times \mathbb{R}^+$, construct a minimal reward machine $\mathcal{H}$ that is consistent with X in that $\mathcal{H}(\lambda) = \rho$ for each $(\lambda, \rho) \in X$.*

To learn minimal consistent reward machines, we adopt a popular approach from classical automata learning (Heule and Verwer 2010; Neider and Jansen 2013; Neider 2014). Our key idea is to generate a sequence of propositional logic formulas φ_k^X for increasing values of $k \in \mathbb{N}\setminus\{0\}$ that satisfy two properties:

- φ_k^X is satisfiable if and only if there exists a reward machine with k states that is consistent with X; and
- a satisfying assignment of the variables in φ_k^X contains sufficient information to derive such a reward machine.

By starting with $k = 1$ and increasing k by one until φ_k^X becomes satisfiable, which we check using a highly-optimized SAT solver, we obtain an effective algorithm that learns a minimal reward machine that is consistent with the given sample. Due to the limited space, however, we have to refer the reader to the extended version of the paper (Xu et al. 2019) for a detailed description of the formulas φ_k^X and the corresponding learning algorithm.

Optimal Convergence

Both tabular q-learning and QRM almost surely converge to an optimal policy. We here show that the same desirable property holds for JIRP. More specifically, in the following sequence of lemmas we show that—given a long enough exploration—JIRP will converge to the reward machine that encodes the reward function of the underlying MDP. We then use this fact to show that the overall learning process converges to an optimal policy (see Theorem 1).

We begin by defining *attainable trajectories*—trajectories that can possibly appear in the exploration of an agent.

Definition 3. *Let $\mathcal{M} = (S, s_I, A, p, R, \gamma, \mathcal{P}, L)$ be a labeled MDP and $m \in \mathbb{N}$ a natural number. We call a trajectory $\zeta = s_0 a_0 s_1 \dots s_k a_k s_{k+1} \in (S \times A)^* \times S$ m-*attainable *if (i) $k \leq m$ and (ii) $p(s_i, a_i, s_{i+1}) > 0$ for each $i \in \{0, \dots, k\}$. Moreover, we say that a trajectory ζ is* attainable *if there exists an $m \in \mathbb{N}$ such that ζ is m-attainable.*

An induction shows that JIRP almost surely explores every attainable trajectory in the limit (i.e., with probability 1 when the number of episodes goes to infinity).

Lemma 1. *Let $m \in \mathbb{N}$ be a natural number. Then, JIRP with $eplength \geq m$ almost surely explores every m-attainable trajectory at least once in the limit.*

Analogous to Definition 3, we call a label sequence $\lambda = \ell_0 \dots \ell_k$ (m-)attainable if there exists an (m-)attainable trajectory $s_0 a_0 s_1 \dots s_k a_k s_{k+1}$ such that $\ell_i = L(s_i, a_i, s_{i+1})$ for each $i \in \{0, \dots, k\}$. An immediate consequence of Lemma 1 is that JIRP almost surely explores every m-attainable label sequence in the limit.

Corollary 1. *JIRP with $eplength \geq m$ almost surely explores every m-attainable label sequence at least once in the limit.*

If JIRP explores sufficiently many m-attainable label sequences for a large enough value of m, it is guaranteed to infer a reward machine that is equivalent to the reward machine encoding the reward function R on all attainable label sequences. This is formalized in the next lemma.

Lemma 2. *Let $\mathcal{M}$ be a labeled MDP and $\mathcal{A}$ the reward machine encoding the reward function of $\mathcal{M}$. Then, JIRP with $eplength \geq 2^{|\mathcal{M}|+1}(|\mathcal{A}|+1) - 1$ almost surely learns a reward machine in the limit that is equivalent to $\mathcal{A}$ on all attainable label sequences.*

Lemma 2 guarantees that JIRP will eventually learn the reward machine encoding the reward function of an underlying MDP. This is the key ingredient for establishing that JIRP almost surely learns an optimal policy in the limit. Note that Lemma 2 also provides a finite bound on the length of the episodes. Therefore, we do not require the length of the episodes to be infinite for guaranteeing optimal convergence, as stated in the following theorem.

Theorem 1. *Let $\mathcal{M}$ be a labeled MDP and $\mathcal{A}$ the reward machine encoding the reward function of $\mathcal{M}$. Then, JIRP with $eplength \geq 2^{|\mathcal{M}|+1}(|\mathcal{A}|+1) - 1$ almost surely converges to an optimal policy in the limit.*

The factor $2^{|\mathcal{M}|+1}$ in the lower bound of $eplength$ arises from our quite general setting in which not every label sequence has to be attainable and the MDP can be nondeterministic. For slightly more restrictive settings, we can provide drastically improved bounds.

Corollary 2. *Let $\mathcal{M}$ and $\mathcal{A}$ be as in Theorem 1. Then:*

- *If $\mathcal{M}$ is deterministic (i.e., $L(s,a,s') = L(s,a,s'')$ implies $s' = s''$), then Theorem 1 is true already for $eplength \geq (|\mathcal{M}|+1)\cdot(|\mathcal{A}|+1) - 1$.*
- *If every label sequence is attainable in $\mathcal{M}$, then Theorem 1 is true already for $eplength \geq 2(|\mathcal{A}|+1) - 1$, which no longer depends on $\mathcal{M}$.*

4 Algorithmic Optimizations

Section 3 provides the base algorithm with theoretical guarantees for convergence to an optimal policy. In this section,

Algorithm 3: JIRP with algorithmic optimizations

1 Initialize the hypothesis reward machine $\mathcal{H}$ with a set of states V
2 Initialize a set of q-functions $Q = \{q^v | v \in V\}$
3 Initialize $X = \emptyset$ and $X_{\text{new}} = \emptyset$
4 **for** *episode* $n = 1, 2, \ldots$ **do**
5 $(\lambda, \rho, Q) = \text{QRM\_episode}(\mathcal{H}, Q)$
6 **if** $\mathcal{H}(\lambda) \neq \rho$ **then**
7 add (λ, ρ) to X_{new}
8 **if** $\big(mod(n, N) = 0$ *and* $X_{new} \neq \emptyset\big)$ **then**
9 $X \leftarrow X \cup X_{\text{new}}$
10 infer $\mathcal{H}_{\text{new}}$ using X
11 $Q_{\text{new}} \leftarrow \text{Transfer}_q(Q, \mathcal{H}, \mathcal{H}_{\text{new}})$
12 $\mathcal{H} \leftarrow \mathcal{H}_{\text{new}}, Q \leftarrow Q_{\text{new}}, X_{\text{new}} \leftarrow \emptyset$

we present an improved algorithm, shown as Algorithm 3, that includes two *algorithmic optimizations*: (1) batching of counterexamples and (2) transfer of q-functions. Both optimizations are designed so that the convergence guarantee of Theorem 1 is preserved.

Batching of Counterexamples

Algorithm 2 infers a new hypothesis reward machine whenever a counterexample is encountered. This could incur a high computational cost in frequently inferring the hypothesis reward machines. In order to adjust the frequency of inferring new reward machines, Algorithm 3 stores each counterexample in a set X_{new}. After each period of N episodes (where $N \in \mathbb{Z}_{>0}$ is a user-defined hyperparameter) and if X_{new} is non-empty, we add X_{new} to the sample X and infer a new hypothesis reward machine $\mathcal{H}_{\text{new}}$ (Lines 8 to 10). Then, Algorithm 3 proceeds with the QRM algorithm for $\mathcal{H}_{\text{new}}$. The procedure repeats until the policy converges.

Transfer of Q-functions

In Algorithm 2, after a new hypothesis reward machine is inferred, the q-functions are re-initialized and the experiences from the previous iteration of RL are not utilized. To utilize experiences in previous iterations, we provide a method to transfer the q-functions from the previously inferred reward machine to the newly inferred reward machine (inspired by the curriculum learning implementation in Icarte et al. (2018)). The transfer of q-functions is based on a notion of *equivalent states*, as defined below.

Definition 4. *Given a reward machine $\mathcal{A}$ and a state $v \in V$, let $\mathcal{A}[v]$ be the machine with v as the initial state. For two reward machines $\mathcal{A}$ and $\hat{\mathcal{A}}$ over the sets V and $\hat{V}$ of states, respectively, two states $v \in V$ and $\hat{v} \in \hat{V}$ are equivalent, denoted by $v \sim \hat{v}$, if and only if $\mathcal{A}[v](\lambda) = \mathcal{A}[\hat{v}](\lambda)$ for all label sequences λ.*

To determine pairwise equivalent states, one can use a simple modification of Hopcroft's partition refinement algorithm (Hopcroft and Ullman 1979). More precisely, if $\mathcal{A}$ has m states and $\hat{\mathcal{A}}$ has n states, one can decide the equivalence of every pair of states in time $\mathcal{O}(2^{\mathcal{P}} \cdot (m+n)^2)$.

Algorithm 4: Transfer_q

1 **Input:** a set of q-functions $Q = \{q^v | v \in V\}$, hypothesis reward machines $\mathcal{H} = (V, v_I, 2^{\mathcal{P}}, \mathbb{R}, \delta, \sigma)$, $\mathcal{H}_{\text{new}} = (V_{\text{new}}, v_{I\,\text{new}}, 2^{\mathcal{P}}, \mathbb{R}, \delta_{\text{new}}, \sigma_{\text{new}})$
2 Initialize $Q_{\text{new}} = \{q_{\text{new}}^{v_{\text{new}}} \mid q_{\text{new}}^{v_{\text{new}}} \in V_{\text{new}}\}$
3 **for** $v_{new} \in V_{new}$, $v \in V$ **do**
4 **if** $v \sim v_{new}$ **then**
5 $q_{\text{new}}^{v_{\text{new}}} \leftarrow q^v$
6 Return Q_{new}

With Definition 4, we provide the following theorem claiming equality of optimal q-functions for equivalent states of two reward machines. We use $q^{*v}(s, a)$ to denote the optimal q-function for state v of a reward machine.

Theorem 2. *Let $\mathcal{A} = (V, v_I, 2^{\mathcal{P}}, \mathbb{R}, \delta, \sigma)$ and $\hat{\mathcal{A}} = (\hat{V}, \hat{v}_I, 2^{\mathcal{P}}, \mathbb{R}, \hat{\delta}, \hat{\sigma})$ be two reward machines encoding the rewards of a labeled MDP $\mathcal{M} = (S, s_I, A, p, R, \gamma, \mathcal{P}, L)$. For states $v \in V$ and $\hat{v} \in \hat{V}$, if $v \sim \hat{v}$, then for every $s \in S$ and $a \in A$, $q^{*v}(s, a) = q^{*\hat{v}}(s, a)$.*

Algorithm 4 shows the procedure to transfer the q-functions between the hypothesis reward machines of consecutive iterations. For any state of the hypothesis reward machine in the current iteration, we check if there exists an equivalent state of the hypothesis reward machine in the previous iteration. If so, the corresponding q-functions are transferred (Line 5). As shown in Theorem 2, the optimal q-functions for two equivalent states are the same.

5 Case Studies

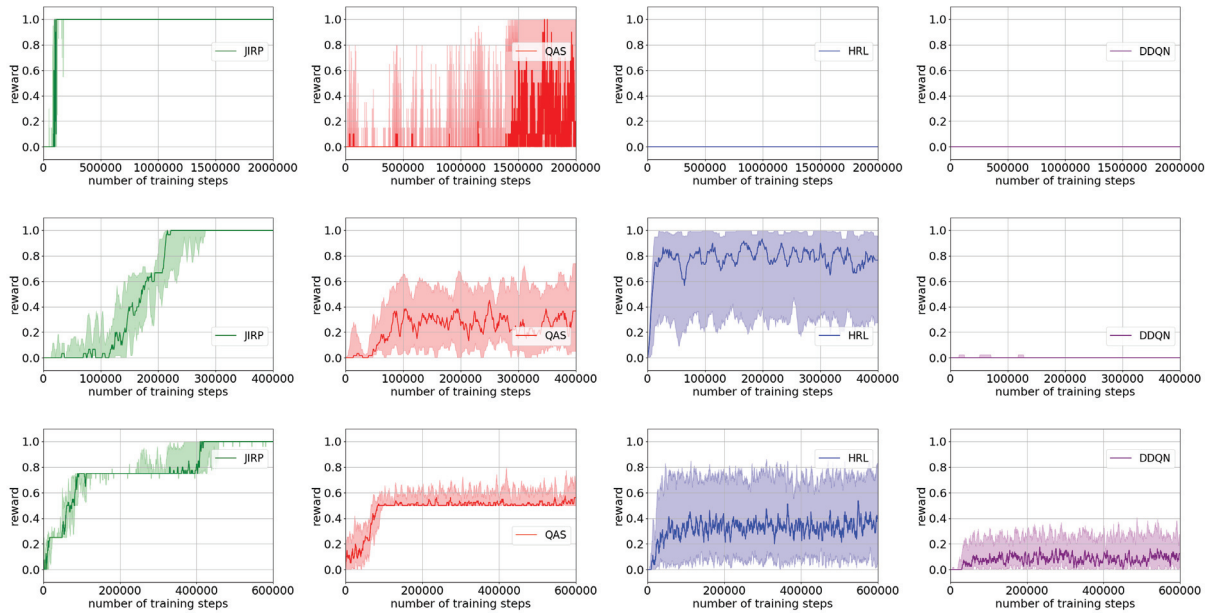
In this section, we apply JIRP to three different scenarios: (1) an autonomous vehicle scenario, (2) an office world scenario adapted from (Icarte et al. 2018), and (3) a Minecraft world scenario adapted from (Andreas, Klein, and Levine 2017). The detailed description of the tasks in the three different scenarios can be found in the extended version (Xu et al. 2019).

We compare the following four different methods:

1) JIRP: We have implemented a prototype of JIRP, which uses the tabular q-learning method (Watkins and Dayan 1992) and the libalf library (Bollig et al. 2010) to infer minimal reward machines.

2) QAS (q-learning in augmented state space): to incorporate the extra information of the labels (i.e., high-level events in the environment), we perform tabular q-learning (Watkins and Dayan 1992) in an augmented state space with an extra binary vector representing whether each label has been encountered or not. We choose the learning rate to be $\alpha = 0.8$ and the discount factor to be $\gamma = 0.9$.

3) HRL (hierarchical reinforcement learning): Following (Kulkarni et al. 2016), we define one option for each propositional variable $p \in \mathcal{P}$, and the option terminates whenever p becomes true.



(a) JIRP (b) QAS (c) HRL (d) DDQN

Figure 3: Attained rewards of 10 independent simulation runs averaged for every 10 training steps for autonomous vehicle scenario (first row), office world scenario (second row), and Minecraft world scenario (third row).

4) DDQN (deep reinforcement learning with double q-learning): we adopt the DDQN method from (Hasselt, Guez, and Silver 2016). The neural network used has 6 fully-connected layers and 64 neurons per layer. The feature inputs of the neural network are the past 200 labels along the trajectory and the current MDP state.

The extended version (Xu et al. 2019) also contains the results of running JIRP with an alternative polynomial-time heuristic for inferring the hypothesis reward machines.

Autonomous Vehicle Scenario

We consider the autonomous vehicle scenario as introduced in the motivating example from Section 1. The set of actions is $A = \{\text{straight, left, right, stay}\}$, corresponding to going straight, turning left, turning right and staying in place. For simplicity, we assume that the labeled MDP is deterministic (i.e, the slip rate is zero for each action).

The set of propositional variables is $\{sp, pr, \mathrm{B}\}$ and the labeling function L is defined by

$$sp \in L(s,a,s') \Leftrightarrow a = \text{stay} \wedge s \in \mathcal{J},$$
$$pr \in L(s,a,s') \Leftrightarrow s'.priority = \top \wedge s \in \mathcal{J},$$
$$\mathrm{B} \in L(s,a,s') \Leftrightarrow s'.x = x_{\mathrm{B}} \wedge s'.y = y_{\mathrm{B}},$$

where $s'.priority$ is a Boolean variable that is true ($\top$) if and only if s' is on the priority roads, $\mathcal{J}$ represents the set of locations where the vehicle is entering an intersection, $s'.x$ and $s'.y$ are the x and y coordinate values at state s, and x_{B} and y_{B} are x and y coordinate values at B (see Figure 1).

We set $eplength = 100$, $N = 100$ (see discussions below for selecting N), and used the transfer of q-functions for the JIRP method. Figure 4 shows the inferred hypothesis reward machine in the last iteration of JIRP in one typical run. The inferred hypothesis reward machine is different from the true reward machine in Figure 2, but these two reward machines are equivalent on all attainable label sequences.

The first row of Figure 3 shows the attained rewards with the four different methods in the autonomous vehicle scenario. JIRP converges to optimal policies within 100,000 training steps, while QAS does not converge to optimal policies, HRL and DDQN are stuck with near-zero cumulative reward for up to two million training steps.

Batch Sizes To test the influence of different batch sizes of counter-examples, we perform JIRP with four different batch sizes of counter-examples: $N = 1$, $N = 10$, $N = 100$ and $N = 1000$. Table 1 shows the average computation time for 10 independent runs with the four different batch sizes in autonomous vehicle scenario. Figure 5 shows the attained median rewards of 10 independent simulation runs with the four different batch sizes in the autonomous vehicle scenario. We observe that with increased batch size, the computation time decreases as the frequency of inferring new hypothesis reward machines decreases (as shown in Table 1). On the other hand, the time to convergence increases when N becomes larger than 100 (as shown in Figure 5), as slow updating of hypothesis reward machines can cause delay for the optimal convergence.

Transfer of Q-functions To test the influence of the transfer of q-functions, we perform JIRP in the autonomous vehicle scenario with and without the transfer of q-functions. As can be seen in Figure 6, JIRP with the transfer of q-functions

Table 1: Average computation time (in seconds) for JIRP and four different batch sizes in autonomous vehicle scenario.

	$N = 1$	$N = 10$	$N = 100$	$N = 1000$
time (s)	3995.99	3193.280	2110.23	1311.79

converges to an optimal policy at about 110,000 training steps, while JIRP without the transfer of q-functions converges to an optimal policy at about 140,000 training steps. Therefore, the transfer of q-functions improves sampling efficiency of JIRP.

Office World Scenario

We consider the office world scenario in the 9×12 gridworld (Icarte et al. 2018). The agent has four possible actions at each time step: move north, move south, move east and move west. After each action, the robot may slip to each of the two adjacent cells with the probability of 0.05, respectively. We use three tasks with different high-level structural relationships among subtasks such as getting coffee, getting mails and going to the office.

We use the same hyperparameters as those in the autonomous vehicle scenario. The second row of Figure 3 shows the cumulative rewards with the four different methods in the office world scenario. JIRP converges to an optimal policy within 150,000 training steps, while QAS and HRL reach only 40% and 80% (respectively) of the optimal median cumulative reward within 400,000 training steps, and DDQN is stuck with near-zero attained reward for up to 400,000 training steps.

Minecraft World Scenario

We consider the Minecraft example in a 21×21 gridworld (Andreas, Klein, and Levine 2017). The four actions and the slip rates are the same as in the office world scenario. We use four tasks including making a plank, making a stick, making a bow and making a bridge.

We use the same hyperparameters as those in the autonomous vehicle scenario. The third row of Figure 3 shows the cumulative rewards with the four different methods in the Minecraft world scenario. JIRP converges to an optimal policy within 600,000 training steps, while QAS, HRL and DDQN reach only 50%, 40% and 20% (respectively) of the optimal median cumulative reward within 600,000 training steps.

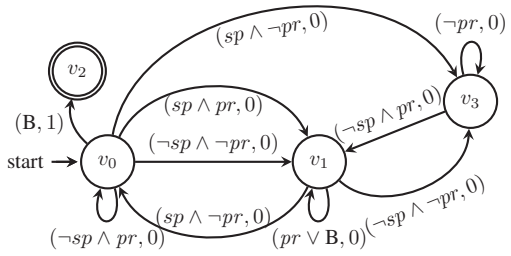


Figure 4: Inferred hypothesis reward machine in the last iteration of JIRP in one typical run in the autonomous vehicle scenario.

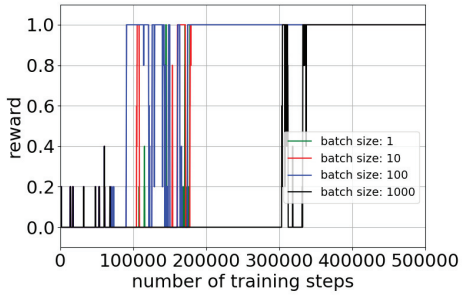


Figure 5: Attained median rewards of 10 independent simulation runs in the autonomous vehicle scenario with four different batch sizes in autonomous vehicle scenario.

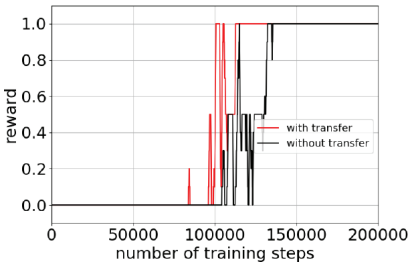


Figure 6: Attained median rewards of 10 independent simulation runs in the autonomous vehicle scenario with and without the transfer of q-functions.

6 Conclusion

We proposed an iterative approach that alternates between reward machine inference and reinforcement learning (RL) for the inferred reward machine. We have proved the optimal convergence of the proposed approach and shown the performance improvement over the baseline methods.

This work opens the door for utilizing automata learning in RL. First, the same methodology can be applied to other forms of RL, such as model-based RL, or actor-critic methods. Second, we will explore methods that can infer the reward machines incrementally (based on inferred reward machines in the previous iteration). Finally, the method to transfer the q-functions between equivalent states of reward machines can be also used for transfer learning between different tasks where the reward functions are encoded by reward machines.

7 Acknowledgments

This work is supported in part by grants DFG 389792660-TRR 248, DFG 434592664, ERC 610150, DARPA D19AP00004, ONR N000141712623, and NASA 80NSSC19K0209.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. 2018. State abstractions for lifelong reinforcement learning. In *ICML'2018*, 10–19.

Akrour, R.; Veiga, F.; Peters, J.; and Neumann, G. 2018. Regularizing reinforcement learning with state abstraction. In *IROS*, 534–539. IEEE.

Aksaray, D.; Jones, A.; Kong, Z.; Schwager, M.; and Belta, C. 2016. Q-learning for robust satisfaction of signal temporal logic specifications. In *IEEE CDC'16*, 6565–6570.

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *AAAI'18*.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *ICML'2017*, 166–175. JMLR. org.

Arora, S., and Doshi, P. 2018. A survey of inverse reinforcement learning: Challenges, methods and progress.

Barto, A. G., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems* 13(1-2):41–77.

Bollig, B.; Katoen, J.; Kern, C.; Leucker, M.; Neider, D.; and Piegdon, D. R. 2010. libalf: The automata learning framework. In *CAV'2010*, 360–364.

Camacho, A.; Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI'2019*, 6065–6073.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.* 13(1):227–303.

Fu, J., and Topcu, U. 2014. Probably approximately correct MDP learning and control with temporal logic constraints. *Robotics: Science and Systems* abs/1404.7073.

Hasselt, H. v.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proc. AAAI'16*, 2094–2100. AAAI Press.

Heule, M., and Verwer, S. 2010. Exact DFA identification using SAT solvers. In *ICGI'2010'*, volume 6339 of *Lecture Notes in Computer Science*, 66–79. Springer.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *ICML'2018*, 2112–2121.

Icarte, R. A. T.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M. P.; and McIlraith, S. A. 2019. Learning reward machines for partially observable reinforcement learning. In *NeurIPS*.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NeurIPS'2016*, 3675–3683.

Li, X.; Vasile, C. I.; and Belta, C. 2017. Reinforcement learning with temporal logic rewards. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots and Syst.*, 3834–3839.

Löding, C.; Madhusudan, P.; and Neider, D. 2016. Abstract learning frameworks for synthesis. In *TACAS'2016*, volume 9636 of *Lecture Notes in Computer Science*, 167–185. Springer.

Nachum, O.; Gu, S.; Lee, H.; and Levine, S. 2018. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 3307–3317.

Neider, D., and Jansen, N. 2013. Regular model checking using solver technologies and automata learning. In *NFM'2013*, volume 7871 of *Lecture Notes in Computer Science*, 16–31. Springer.

Neider, D. 2014. *Applications of automata learning in verification and synthesis*. Ph.D. Dissertation, RWTH Aachen University.

Oncina, J., and Garcia, P. 1992. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, 49–61. World Scientific.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, 1043–1049.

Shallit, J. O. 2008. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112(1-2):181–211.

Taylor, M. E., and Stone, P. 2007. Cross-domain transfer for reinforcement learning. In *Proc. ICML'07*, 879–886. New York, NY, USA: ACM.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2018. Teaching multiple tasks to an RL agent using LTL. In *Proc. AAMAS'2018*, 452–461.

Watkins, C. J. C. H., and Dayan, P. 1992. Q-learning. *Machine Learning* 8(3):279–292.

Wen, M.; Papusha, I.; and Topcu, U. 2017. Learning from demonstrations with high-level side information. In *Proc. IJCAI'17*, 3055–3061.

Wulfmeier, M.; Ondruska, P.; and Posner, I. 2015. Maximum entropy deep inverse reinforcement learning. In *NeurIPS, Deep Reinforcement Learning Workshop*, volume abs/1507.04888.

Xu, Z., and Topcu, U. 2019. Transfer of temporal logic formulas in reinforcement learning. In *Proc. IJCAI'2019*, 4010–4018.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2019. Joint inference of reward machines and policies for reinforcement learning. *CoRR* abs/1909.05912.